I. PROCESS API

Execution Model – Assembler (simplified)

OS interacts directly with compiled programs
- switch between processes/threads ⇝ **save**/**restore** state
- deal with/pass on **signals**/**exceptions**
- receive **requests** from applications

Instructions:
- `mov`: Copy referenced data from second operand to first operand
- `add`/`sub`/`mul`/`div`: Add,... from second operand to first operand
- `inc`/`dec`: increment/decrement register/memory location
- `shl`/`shr`: shift first operand left/right by amount given by second operand
- `and`/`or`/`xor`: calculate bitwise and,... of two operands storing result in first
- `not`: bitwise negate operand

Execution Model – Stack (x86)

stack pointer (SP): holds address of stack top (growing downwards)

stack frames: larger stack chunks

base pointer (BP): used to organize stack frames

Execution Model – jump/branch/call commands (x86)

`jmp`: continue execution at operand address

`j$condition`: jump depending on PSW content
true ⇝ jump
false ⇝ continue
examples: `je` (jump equal), `jz` (jump zero)

`call`: push function to stack and jump to it

`return`: return from function (jump to return address)

Execution Model – Application Binary Interface (ABI)

standardizes binary interface between programs, modules, OS:
- executable/object file layout
- calling conventions
- alignment rules

calling conventions: standardize exact way function calls are implemented
⇝ interoperability between compilers

Execution Model – calling conventions (x86)

function call (caller):

1. save local scope state
2. set up parameters where function can find them
3. transfer control flow

function call (called function):

1. set up new local scope (local variables)
2. perform duty
3. put return value where caller can find it
4. jump back to caller (IP)